NIST GCR 11-949

Representation of functional networks in STEP AP210 with application to SPICE circuit simulation

James Stori
SFM Technology, Inc.,

National Institute of Standards and Technology
U.S. Department of Commerce

NIST GCR 11-949

Representation of functional networks in STEP AP210 with application to SPICE circuit simulation

Prepared for

U.S. Department of Commerce

Information Technology Laboratory

National Institute of Standards and Technology

Gaithersburg, MD 20899-XXXX

By

James Stori
SFM Technology, Inc.,

November, 2011

U.S. Department of Commerce
John Bryson, Secretary

National Institute of Standards and Technology
Patrick D. Gallagher, Director

Representation of functional networks in STEP AP210 with application to SPICE circuit simulation

STEP AP210 (ISO 10303-210) supports a comprehensive functional network model representation. SPICE (Simulation Program with Integrated Circuit Emphasis) is a general purpose, industry standard, analog electronic circuit simulation. Originally developed at the University of California, Berkeley in the 1970s,[1,2] SPICE has evolved into a de-facto standard for analog circuit simulation with several prominent variants, both open source and commercial, including SPICE3 from UC Berkeley[3], PSPICE[4] (now owned by Cadence, HSPICE[5] (now owned by Synopsis), and LTspice[6] from Linear Technology.

This document provides an overview of the functional network representation in AP210 at both the ARM (application reference model) and MIM (module integration model) levels, and proposes a recommended practice for bi-directional mapping between a SPICE circuit representation and an AP210 functional network representation.

Notation

In the discussion that follows, ARM application objects will be written with the leading character capitalized (i.e. Functional_unit), while MIM entities will be written in all lower case (component_functional_unit).

Overview of the AP210 Functional Network Model

The top-level ARM AOs and relationships involved in the definition of a functional network are described in Figure 1, below. A Functional_unit_network_definition is a description of a version (Functional_version) of a product (Functional_product) through a network-based representation. A Functional_unit_network_definition consists of a series of network nodes (Functional_unit_network_node_definition) that reference the Functional_unit_network_definition. A network definition is required to have a usage view (Functional_unit_usage_view), which defines the externally visible interface of an instance (Functional_unit) of the functional product. Typically, a subset of the nodes of a network definition will be mapped to nodes (Functional_unit_usage_view_terminal_definition) in the usage view. The terminal definition represents an individual signal of a usage view. Most commonly, the terminal definition will be of the subtype Scalar_terminal_definition, and the signal_name attribute will be the known identifier for the terminal. A top-level network representation is composed of instances of either other networks or base network elements interconnected through the nodes of the network. Each instance that is used to compose the

[1] Nagel, L. W, and Pederson, D. O., *SPICE (Simulation Program with Integrated Circuit Emphasis)*, Memorandum No. ERL-M382, University of California, Berkeley, Apr. 1973

[2] Nagel, Laurence W., *SPICE2: A Computer Program to Simulate Semiconductor Circuits*, Memorandum No. ERL-M520, University of California, Berkeley, May 1975

[3] http://bwrc.eecs.berkeley.edu/classes/icbook/spice/

[4] http://www.cadence.com/products/orcad/pspice_simulation/pages/default.aspx

[5] http://www.synopsys.com/Tools/Verification/AMSVerification/CircuitSimulation/HSPICE/Pages/default.aspx

[6] http://www.linear.com/designtools/software/

network definition is a Functional_unit, regardless of whether it is itself a network definition, or a base network element. Each node in the network will be connected to at least one terminal (Functional_unit_terminal) of a Functional_unit through a Functional_unit_terminal_node_assignment.

The key ARM application objects and relationships relevant for the description of a Functional_unit are detailed in Figure 2. A Functional_unit is an instance, or occurrence, of a functional product, whose interface is provided through its usage view. The Functional_unit is 'defined by' a Functional_unit_usage view. Each Functional_unit will have a Functional_unit_terminal for each terminal definition in the usage view, and must directly reference the corresponding Functional_unit_usage_view_terminal_definition.

Very commonly, parametric data must be associated with either a Functional_product or an individual instance of a functional product, a Functional_unit. A Model_parameter is used to represent the underlying parameter or variable. A specific value may be assigned to the model parameter and associated with either the Functional_product or Functional_unit through a Parameter_assignment, as shown in Figure 2. It is also possible for a specific value to the assigned to the parameter by the Functional_product, and then overridden by a specific Functional_unit. In this case, the Parameter_assignment_override should be employed. Groupings and associations among model parameters and functional products can be achieved through a Class_with_attributes. As will be discussed below, the recommended practice for the SPICE representation will employ this mechanism in the representation of SPICE circuit elements and models.

A particular type of polarized capacitor, for example, may be a recurring element in a circuit network. This polarized capacitor would be a functional product. Its usage view would have two terminals, identified as the 'anode' and 'cathode'. Each instance of this type of capacitor would be a Functional_unit. At the discretion of the modeling application and context, characteristics of the functional model, such as the equivalent series resistance, equivalent series inductance, nominal capacitance, etc., might be associated with either the Functional_unit, the Functional_product, or both (a common value associated with the Functional_product, and selectively overridden by the Functional_unit in specific cases.

The MIM mapping of the key ARM AOs and relationships in the functional network model representation are detailed in Figures 3 and 4. In these figures, the mapped ARM AO is selectively provided in blue to aid in the interpretation of the mapping. With reference to Figure 3, note that both the Functional_unit_usage_view and the Functional_unit_network_definition both map to the MIM entity functional_unit. The name attribute of the associated product_definition_context ('functional design usage' and 'functional network design', respectively, may be used to identify the two AO concepts, as may the directionality of the product_definition_relationship (with name 'design usage') that relates the two. The Functional_product maps to a MIM product entity, with the associated product_context and product_related_product_category attributes outlined in the figure. Certain aspects of the terminology in the ARM to MIM mapping may lead to confusion. Particularly noteworthy is that the ARM AO Functional_unit_definition (either network definition or usage view) maps to the MIM entity functional_unit, while the ARM Functional_unit (instance of a Functional_unit_definition) maps to the MIM component_functional_unit.

The remaining AP210 concepts needed to support the mapping and representation of a functional network will be discussed below in the specific context of the SPICE circuit network representation. A representative example will be used throughout the discussion to illustrate the concepts and mappings.

Due to its position as a leading freely available implementation of SPICE, LTspice IV has been used for the examples and proof-of-concept JAVA implementation of the bi-directional SPICE<->AP210 mapping. However, the illustrated concepts and examples should be readily transferable to alternate SPICE implementations with minor modification.

SPICE circuit

Figure 5 contains both a schematic and netlist representation of a SPICE circuit provided with the LTspice IV installation. As this recommended practice is limited in scope to the functional network only, the netlist representation shall serve as the interface to the SPICE circuit. This particular network consists of 5 nodes and 8 circuit elements interconnecting the nodes. Circuit elements are instances of functional building blocks of the simulated circuit such as resistors, capacitors, diodes, transistors. The specific set of circuit elements may vary among SPICE implementations. The nodes are identified by unique string identifiers within a circuit. All SPICE circuits have an explicit ground node named ‘0’. This circuit has 4 additional nodes, with string identifiers ‘N001’, ‘N002’, ‘N003’, and ‘N004’. The top-level SPICE circuit is represented by a functional_unit (Functional_unit_network_definition. Each of the network nodes will be represented as a network_node_definition (Functional_unit_network_node_definition) whose id attribute stores the string identifier of the SPICE node reference. All nodes of a top-level SPICE circuit may be monitored or interrogated by the end-user during a simulation. For this reason, the usage view of the circuit representation will contain all of the nodes in the network definition. The AP210 representation enables explicit management of versions of circuit models through the Functional_version and Functional_product AOs. There is no corresponding formal mechanism for configuration management within the SPICE representation.

SPICE circuit element

Each of the individual circuit elements will map to a Functional_unit (component_functional_unit) in the AP210 model. Each circuit element is an instance of an underlying SPICE model element, typically with one or more parameter values explicitly specified. Circuit elements may or may not include a reference to a separate .model declaration which enables grouping and reuse of parameter values associated with a common element. The syntax of a SPICE element includes the instance identifier and sequence dependent references to the network nodes for the terminals of the element, followed by the parameters prescribed to the element. The leading letter of the instance identifier determines the underlying model reference.

For example, the syntax of the inductor element (from the LTspice users guide) is given as follows:

```
Syntax: Lxxx n+ n- <inductance> [ic=<value>]
+ [Rser=<value>] [Rpar=<value>]
+ [Cpar=<value>] [m=<value>] [temp=<value>]
```

In the above, values in brackets are optional. Element L4 in Figure 5 is an instance of an inductor between nodes ‘N003’ and ground (‘0’) represented as follows:

```
L4 N003 0 1mH Rser=100
```

L4 has an inductance of 1mH, and one of the optional parameter values (Rser) is provided. In the AP210 functional representation, an individual circuit element instance such as L4 is a Functional_unit (component_functional_unit) which is an instance of a functional product. The recommended practice for representation of SPICE circuit elements in AP210 is to create a functional product for each SPICE model that is to be used in the circuit. This functional product will have a series of model parameters to which specific values can be assigned. With reference to Figure 2, specific values can be assigned to a model parameter at either the level of the functional product or the instance (Functional_unit). In the case of a SPICE element that does not have a .model reference, all parameters associated with the specific circuit element instance will be populated through a Parameter_assignment of the functional_property aggregate of the Functional_unit.

Although units are implicit in the SPICE netlist, all parameter values are to be populated with explicit units in the AP210 representation. Additionally, it is recommended that model parameters be explicitly associated with their respective Functional_product. As shown in Figure 2, the Class_with_attributes can be used as a grouping mechanism for this purpose. Each model parameter will be directly related to this Class_with_attributes through a Classification_attribute.

In the case of the inductor, a Functional_product would be created for the SPICE inductor model, as well as a corresponding Class_with_attributes. Each of the parameters for the specific model, including the inductance value, equivalent series and parallel resistance values (Rser, Rpar), etc. would be represented as a Model_parameter associated with the Class_with_attributes. Figure 6 details both the ARM AOs and relationships and corresponding MIM mappings for these concepts.

The model parameter is only meaningful in the context of the specific simulation implementation in which it is defined. For this reason, the model parameters should provide a reference to the external system in which they are applied. The reference_document derived attribute on the Model_parameter is used to access this external reference document. Figure 6 shows the ARM AOs and MIM entities appropriate for providing a reference to an external digital document that may be accessed through a URL. The External_source_identification (applied_external_identification_assignment) provides an external reference for a Digital_document_definition. The identification_role identifies the protocol for interpreting the external_source, in this case, a URL, while the source_id attribute contains the specific URL. In the context of a LTspice model, it would be appropriate to explicitly reference the current version of the user’s guide in which the model parameters are based. It is also recommended that the same referenced digital document be associated with the characterized_class (Class_with_attributes).

Figure 7 details the ARM application objects and relationships used in the representation of a Parameter_assignment . The Numerical_item_with_unit contains an explicit declaration of the unit of the represented value, which may be a fundamental unit subtype such as a Time_Unit or a Derived_unit composed of multiple Derived_unit_element. Figures 8 and 9 contain the corresponding MIM mapping of the concepts relevant in the expression of parametric data, and its relationship to the functional model. With reference to Figure 8, note that a single parameter_assignment entity serves as both the Parameter_assignment and the Representation application objects from the ARM.

In the top portion of Figure 8, a parameter_assignment is associated with a component_functional_unit (Functional_unit). In Figure 9, a parameter_assignment is associated with a product (Functional_product) through a product_specific_parameter_value_assignment. In the event that a parameter value is assigned to a model parameter for a functional_product, and then overridden by another parameter value associated with a specific component_functional_unit, a parameter_assignment_override would be employed. In the recommended practice for SPICE model representation, this will not happen, as parameters values are associated with either the component_functional_unit or the functional_product, but not both.

The MIM type hierarchy involved in the expression of measure values and units is somewhat involved. The representation_item corresponding to a Numerical_item_with_unit will be a complex of a measure_representation_item and a measure_with_unit. If the measure corresponds to an available subtype of measure_with_unit, such as time_measure_with_unit in the example of Figure 8, this more specific subtype should be used (i.e. measure_representation_item+time_measure_with_unit). A similar concept applies to the representation of units. The SPICE parameters are expressed in either base SI units or derived SI units. Figures 8 and 9 illustrate both cases. In the event of a base SI unit, a complex of si_unit with the specific subtype representing the unit of measure (i.e. electric_current_unit or time_unit) should be populated. Figure 9 illustrates a complete example of the population of a parameter value expressed in a derived SI unit (in this case V/deg. C) associated with a model_parameter and a product (Functional_product).

SPICE .model

Certain circuit elements have many parameters. These elements require a reference to a .model declaration where a set of common parameter values are prescribed. Consider the JFET transistor model whose syntax is as follows:

```
Syntax: Jxxx D G S <model> [area] [off] [IC=Vds, Vgs] [temp=T]
```

The transistor has three terminals identified by their specification sequence. The <model> entry represents a reference to a .model declaration, which is required. The remaining bracketed entries are optionally specified parameters.

An example taken from the circuit of Figure 5 is JFET transistor J1, and the corresponding .model 2N5484. The netlist representation is as follows:

```
J1 N001 N002 N003 2N5484
…
.model 2N5484 NJF(Is=.25p Alpha=1e-4 Vk=80 Vto=-1.5 Vtotc=-3m Beta=3.0m
Lambda=10m Betatce=-.5 Rd=10 Rs=10 Cgs=4p Cgd=4p Kf=3e-17 mfg=Siliconix)
```

In this particular example, all of the parameter are specified through the .model reference. For circuit elements with a required .model reference, the recommended practice is that the .model be populated as a Functional_product, and with Parameter_assignment for each of the parameter values provided in the .model declaration. The recommended population for the key model identification strings is as follows: The .id attribute contains the identifier of the model (2N5484) while the .name attribute contains the explicit model identifier (NJF). Finally, the associated Class_with_attributes (characterized_class) stores the circuit element type 'J' as its id. The individual circuit element J1 is then a Functional_unit (component_functional_unit) referencing the usage view of this functional product. If any of the optional parameter values had been specified for element J1, each would be populated through a parameter_assignment associated with the component_functional_unit. Figure 9 illustrates the population of the Vtotc parameter of the 2N5484 model. The AP210 functional model representation enables formal declaration of the model parameters, explicit support for units, and re-use through the Functional_product concept.

SPICE .subck

A .subck declaration enables the definition of a reusable circuit element. Multiple instances of a subcircuit can be referenced in a single circuit, and subcircuits can be declared within other subcircuits to arbitrary degrees of complexity. A subcircuit maps to a Functional_unit_network_definition. In the AP210 representation, there is no conceptual difference between a top-level circuit network definition, and a reusable network definition corresponding to a subcircuit. The AP210 functional model has the benefit of formalizing the notion of the interface to a functional unit definition through the usage view. Unlike the top-level network definition, a subcircuit has an implicit interface which is that subset of the nodes of the subcircuit named in the .subck declaration. As in the other circuit elements, the subcircuit element (X) relies on the sequence of the node references to map to the interface nodes of the subcircuit. A simple example of a subcircuit declaration (from the LTspice users guide) is provided below:

```
*
* This is the circuit definition
X1 a b 0 divider
V1 a 0 10
C2 N004 0 50p
C3 N003 0 750p
X2 N003 b N004 divider
* this is the definition of the subcircuit
.subckt divider x1 x2 x3
r1 x1 x2 2k
r2 x2 x3 2k
.ends
.tran 3µ
.end
```

In the above, example, the subcircuit 'divider' is populated as a Functional_product (product) with both a Functional_unit_network_defintion and Functional_unit_usage_view (each a functional_unit). The network definition contains three nodes (network_node_definition) In this case all three nodes of the subcircuit network are exposed to the subcircuit interface, so the usage view contains these same three nodes. The two resistors within the subcircuit are each a Functional_unit (component_functional_unit) of a common resistor model. The terminals of these resistor instances are associated with the nodes of the subcircuit, and are not visible to the top-level circuit network representation. ‘X1’ is a subcircuit element – an instance of the divider subcircuit. The terminals of the X1 subcircuit element are mapped to the corresponding nodes in the interface of the divider declaration based on their sequence (a->x1, b->x2, 0->x3). In the AP210 functional representation, X1 will be populated as a Functional_unit (component_functional_unit) referencing the usage view of the divider subcircuit. X1 will have three Functional_unit_terminals (component_functional_terminal) each associated with a corresponding terminal (Scalar_terminal_definition) in the usage view of the Functional_product corresponding to the subcircuit.

Simulator Directives (Dot Commands)

In addition to the definition of a circuit network, consisting of the declaration of circuit elements, SPICE models, and subcircuits as discussed above, the SPICE netlist representation will typically include a variety of simulator directives that specify the type of analysis to be performed as well as a variety of details concerning the simulation execution. As these directives do not contribute to the circuit topology, they are not included in the AP210 functional model representation.

Case Study and Validation

To validate and demonstrate the proposed bi-directional mapping, a prototype implementation has been developed in JAVA based on the JSDAI[7] library for manipulating EXPRESS data models. The JAVA implementation supports a subset of the most common LTspice IV circuit elements and parameters, and may be readily extended to support other SPICE implementations and models. The JAVA implementation parses the original SPICE circuit and populates an internal Java representation of the SPICE network, circuit elements, and models. This SPICE network is then mapped to the corresponding AP210 representation and written out as a Part 21 physical file. The process is the repeated in the inverse direction with the AP210 model of the functional network read and converted to the corresponding SPICE model. The equivalence of the input and output SPICE networks has been verified by using both to drive the simulation, and comparing waveforms of representative traces from each.

To support round-trip verification of the implementation, simulator directives are extracted from the input netlist and appended to the output netlist – a comment to this effect is included in the output netlist files indicating the start of the appended dot directives. These directives are not persisted in the AP210 functional model. All other data, including the complete circuit representation including all parametric and model details in the output files has been extracted directly from the AP210 functional representation.

Four representative example circuits, provided with the LTspice IV installation have been used to validate and demonstrate the bi-directional mapping. These examples contain a wide variety of circuit elements, models, and parameter representations. The SPICE netlists generated from the AP210 functional network models for these representative circuits are contained in the Appendix. The original

[7] http://www.jsdai.net/

netlists, along with the generated AP210 functional models, and corresponding output netlists are provided in the demo files directory of the prototype bi-directional SPICE<->functional network JAVA / JSDAI implementation.

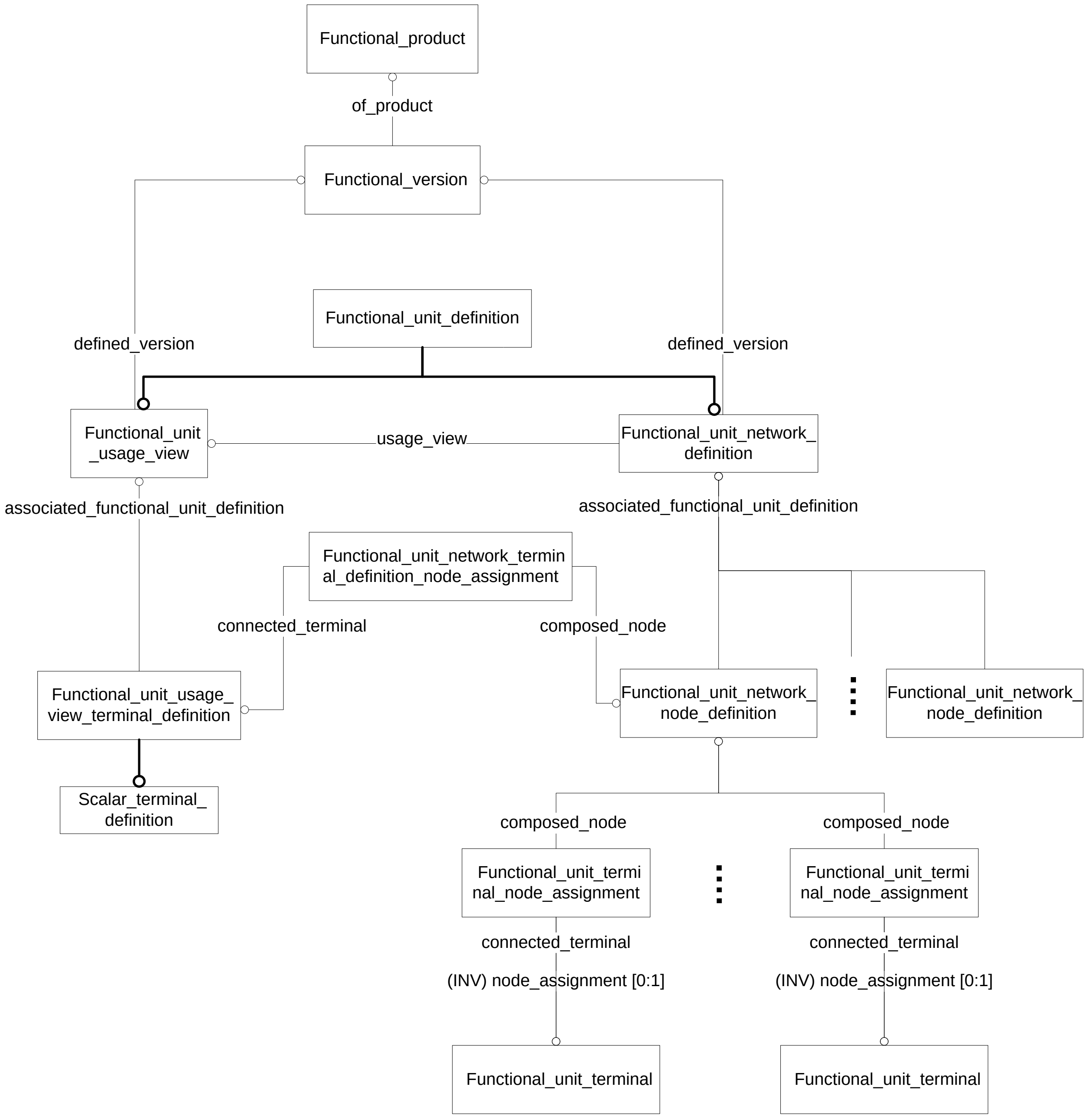

Functional_product
of_product
Functional_version
defined_version
Functional_unit_definition
defined_version
Functional_unit _usage_view
usage_view
Functional_unit_network_ definition
associated_functional_unit_definition
associated_functional_unit_definition
Functional_unit_network_termin al_definition_node_assignment
connected_terminal
composed_node
Functional_unit_usage_ view_terminal_definition
Functional_unit_network_ node_definition
Functional_unit_network_ node_definition
Scalar_terminal_ definition
composed_node
composed_node
Functional_unit_termi nal_node_assignment
Functional_unit_termi nal_node_assignment
connected_terminal
connected_terminal
(INV) node_assignment [0:1]
(INV) node_assignment [0:1]
Functional_unit_terminal
Functional_unit_terminal

Figure 1. Top-level ARM application objects in the definition of a functional network.

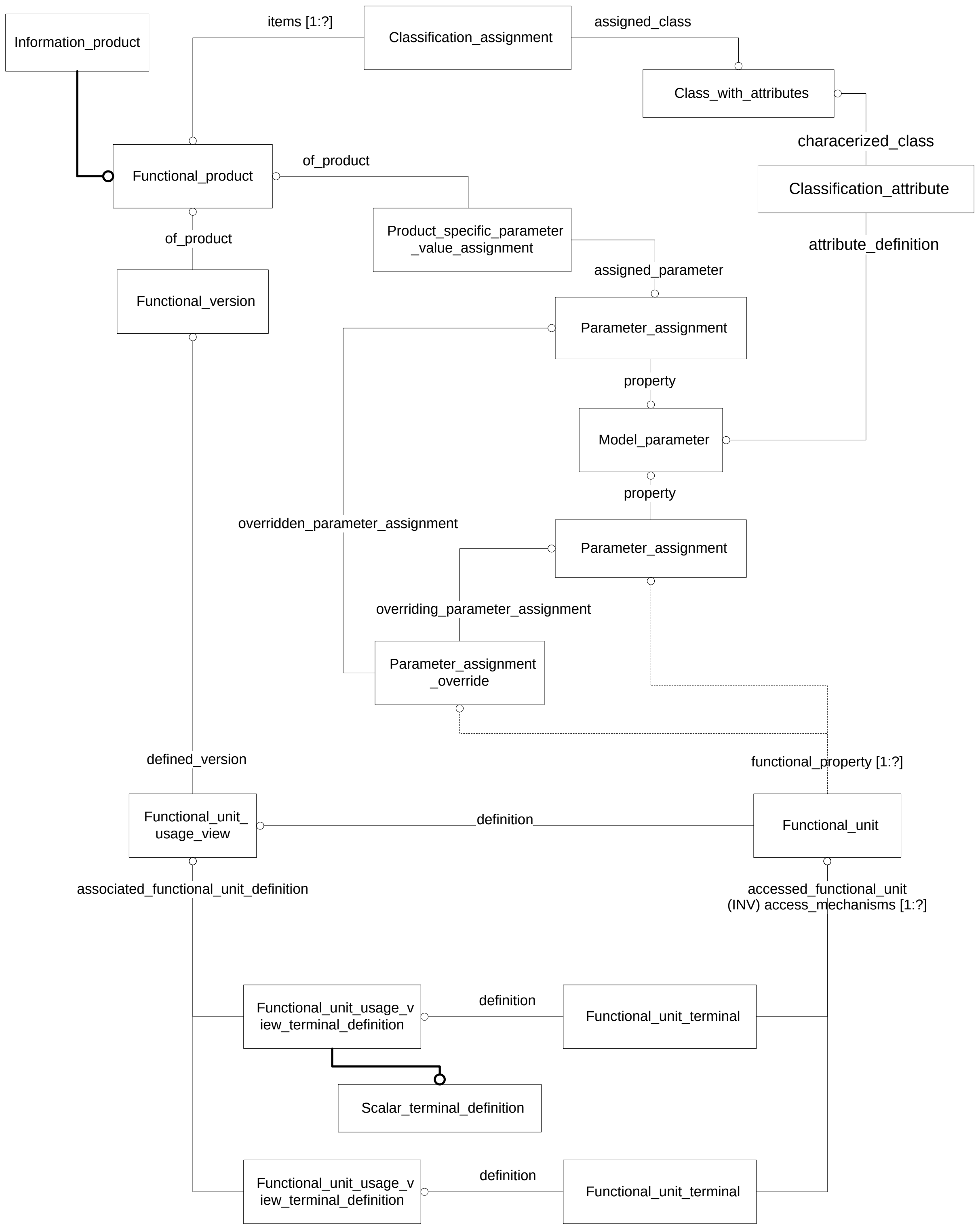


Figure 2. Top-level ARM application objects in the description of a Functional_unit.

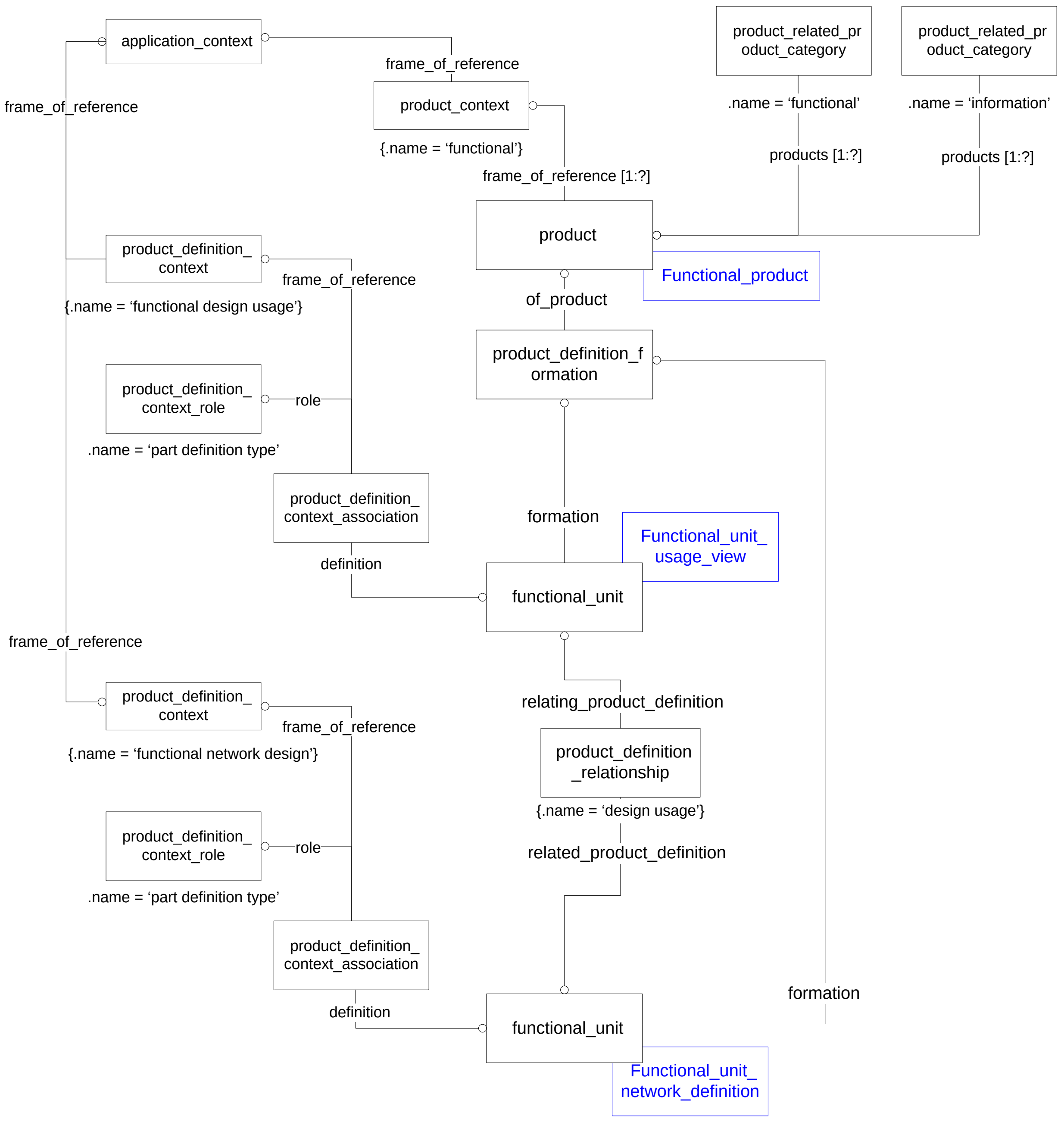

application_context
frame_of_reference
product_context
{.name = 'functional'}
frame_of_reference [1:?]
product_related_product_category
.name = 'functional'
products [1:?]
product_related_product_category
.name = 'information'
products [1:?]
frame_of_reference
product
Functional_product
product_definition_context
frame_of_reference
{.name = 'functional design usage'}
of_product
product_definition_formation
product_definition_context_role
role
.name = 'part definition type'
product_definition_context_association
formation
Functional_unit_usage_view
definition
functional_unit
frame_of_reference
product_definition_context
frame_of_reference
{.name = 'functional network design'}
relating_product_definition
product_definition_relationship
{.name = 'design usage'}
related_product_definition
product_definition_context_role
role
.name = 'part definition type'
product_definition_context_association
definition
functional_unit
formation
Functional_unit_network_definition

Figure 3. MIM mapping of AOs related to the Functional_product and Functional_unit_definition.

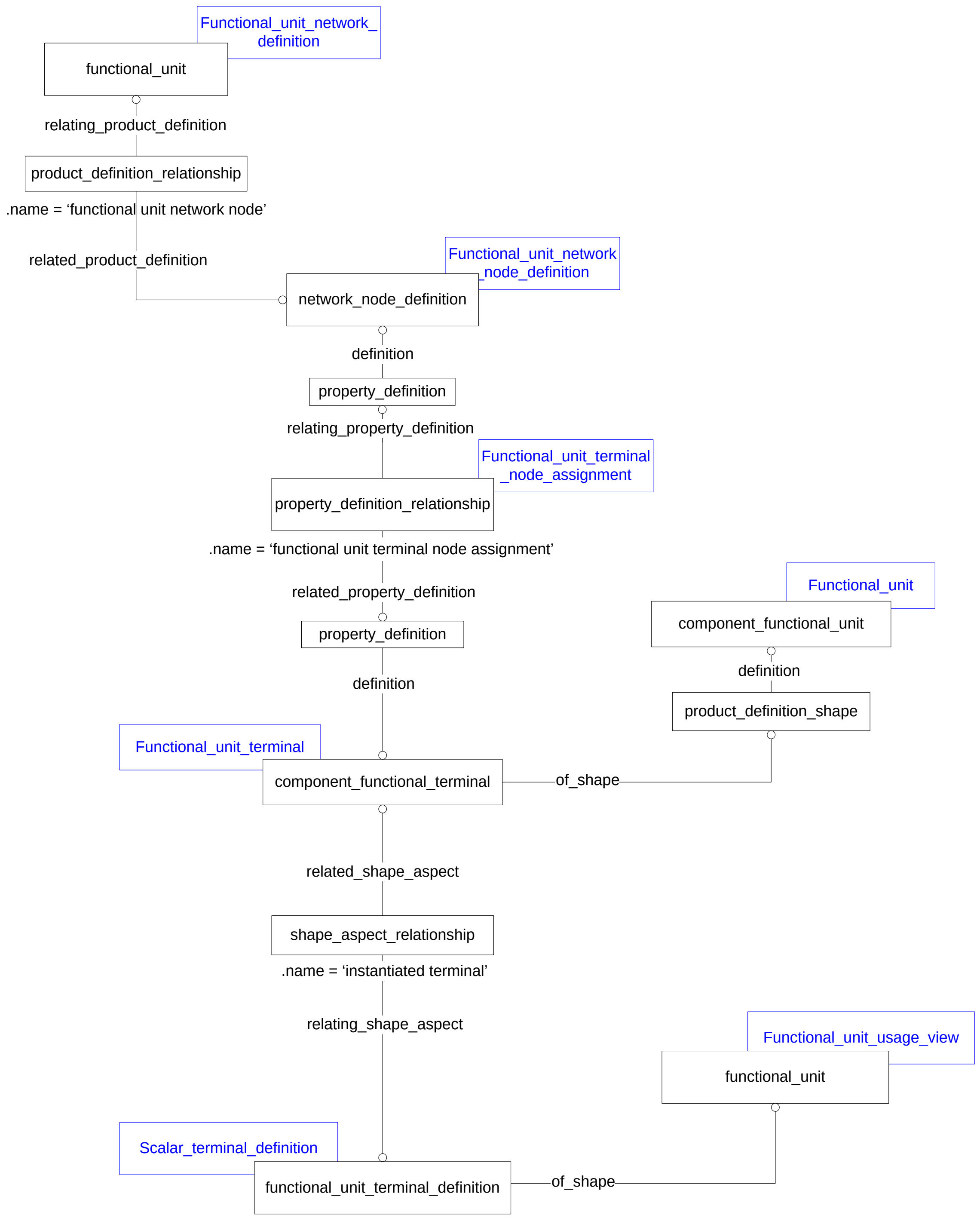
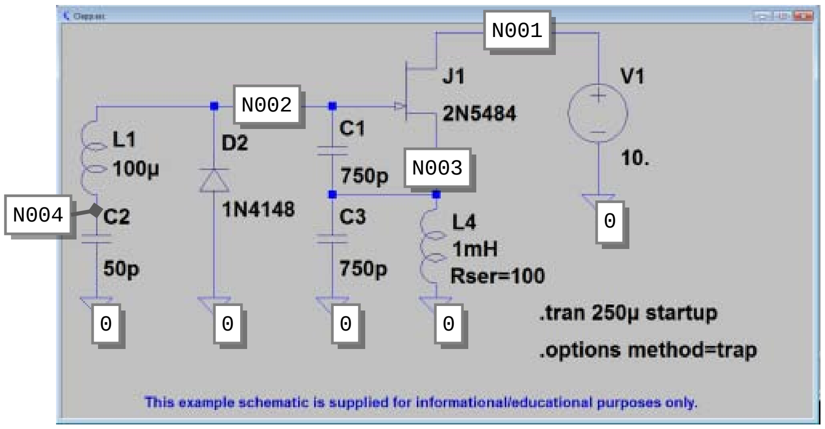

Functional_unit_network_definition
functional_unit
relating_product_definition
product_definition_relationship
.name = ‘functional unit network node’
related_product_definition
Functional_unit_network_node_definition
network_node_definition
definition
property_definition
relating_property_definition
Functional_unit_terminal_node_assignment
property_definition_relationship
.name = ‘functional unit terminal node assignment’
related_property_definition
property_definition
definition
Functional_unit
component_functional_unit
definition
product_definition_shape
Functional_unit_terminal
component_functional_terminal
of_shape
related_shape_aspect
shape_aspect_relationship
.name = ‘instantiated terminal’
relating_shape_aspect
Functional_unit_usage_view
functional_unit
Scalar_terminal_definition
functional_unit_terminal_definition
of_shape

Figure 4. MIM mapping of AOs related to the Functional_unit and Functional_unit_network_definition.



```
* Clapp.net extracted from C:\Program Files
(x86)\LTC\LTspiceIV\examples\Educational\Clapp.asc
J1 N001 N002 N003 2N5484
V1 N001 0 10.
L4 N003 0 1mH Rser=100
C1 N002 N003 750p
C2 N004 0 50p
C3 N003 0 750p
L1 N002 N004 100µ
D2 0 N002 1N4148
.model 1N4148 D(Is=2.52n Rs=.568 N=1.752 Cjo=4p M=.4 tt=20n Iave=200m Vpk=75
mfg=OnSemi type=silicon)
.model 2N5484 NJF(Is=.25p Alpha=1e-4 Vk=80 Vto=-1.5 Vtotc=-3m Beta=3.0m
Lambda=10m Betatce=-.5 Rd=10 Rs=10 Cgs=4p Cgd=4p Kf=3e-17 mfg=Siliconix)
.model NJF NJF
.model PJF PJF
.tran 250µ startup
.options method=trap
* This example schematic is supplied for informational/educational purposes
only.
.backanno
.end
```

Figure 5. A representative network (Clapp.asc) provided with the LTspice IV installation to illustrate the representation and mapping concepts.

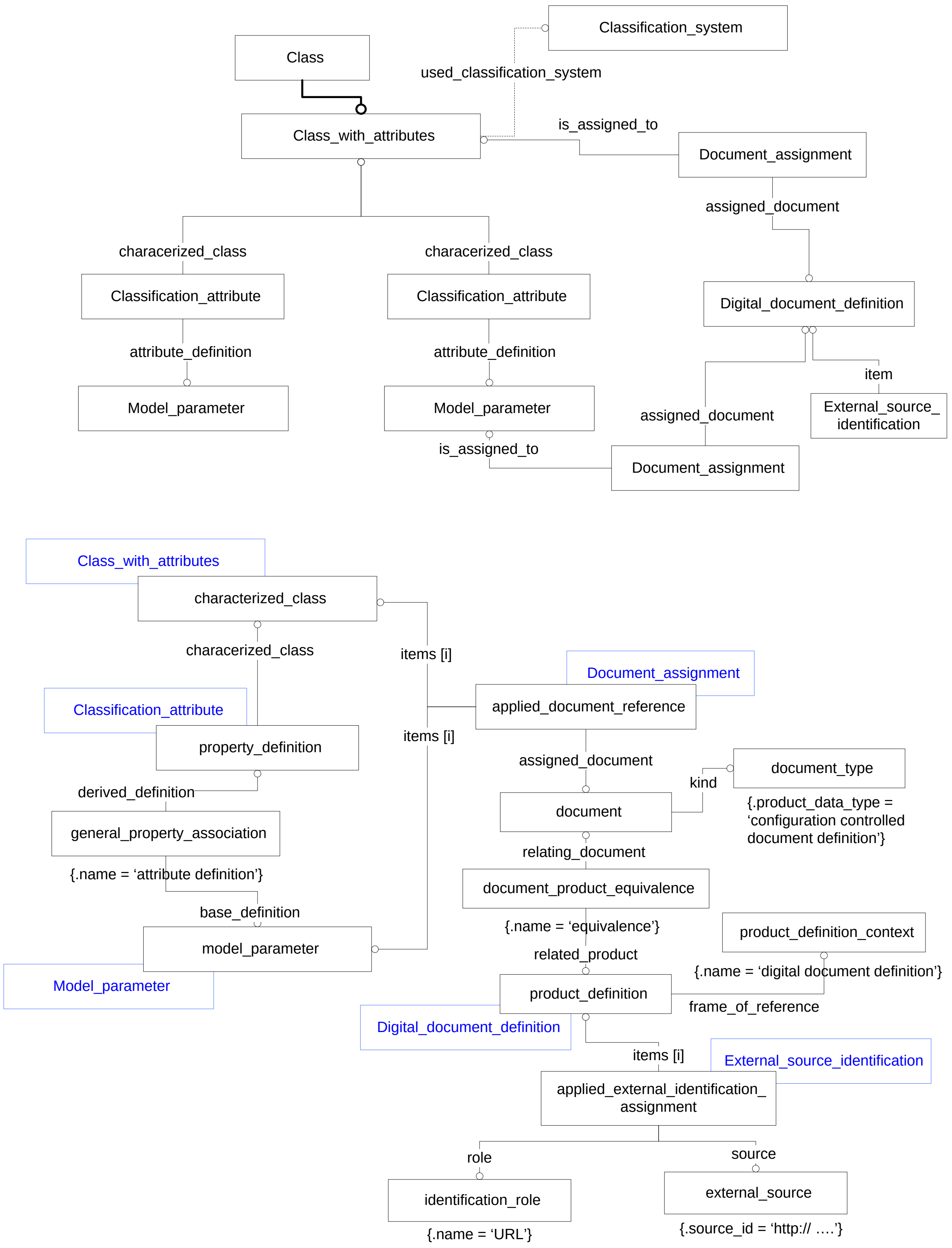


Figure 6. ARM AOs (top) and corresponding MIM mappings (below) for the grouping and association of model parameters.

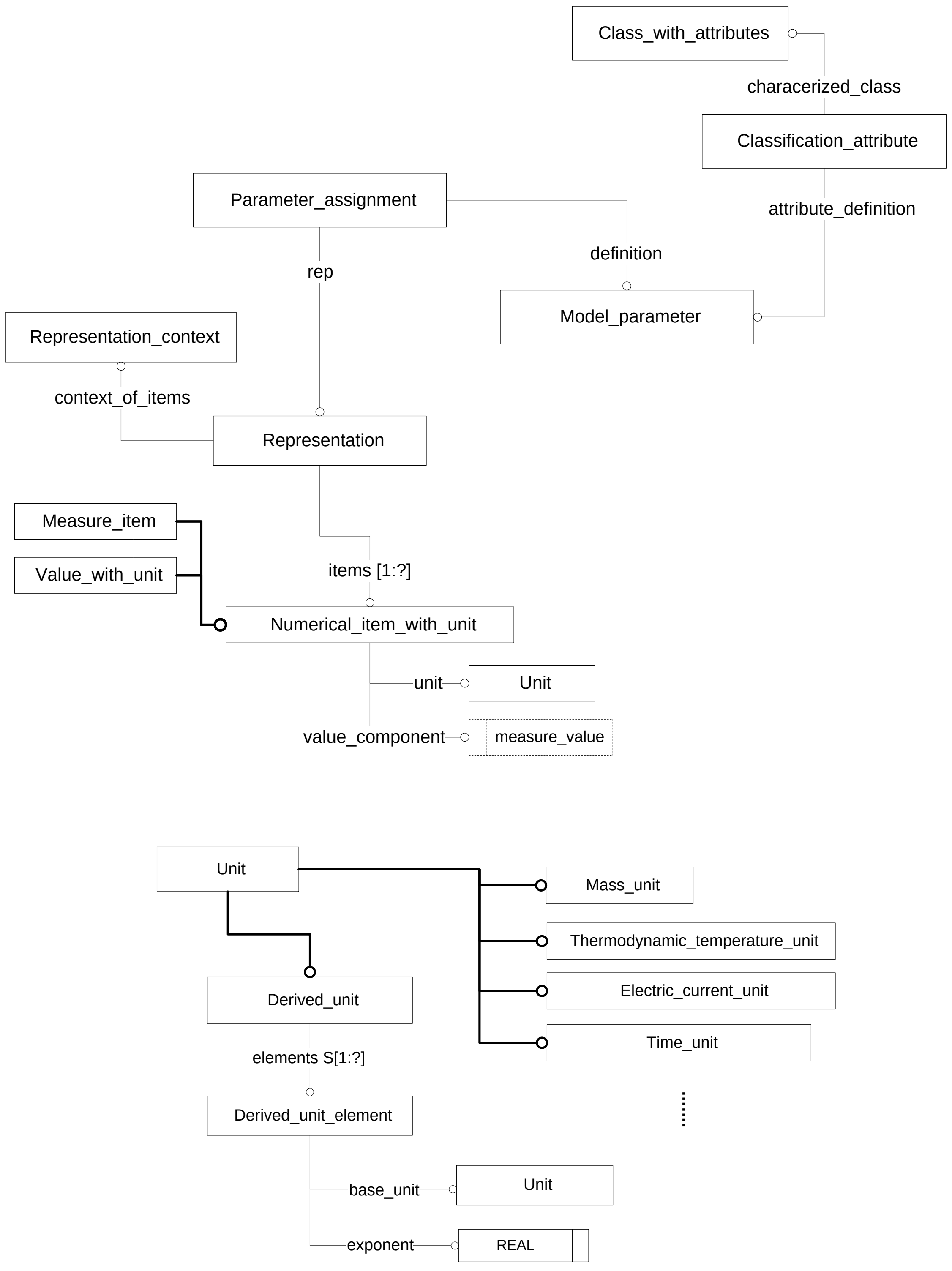


Figure 7. ARM AOs and relationships involved in the representation of a Parameter_assignment.

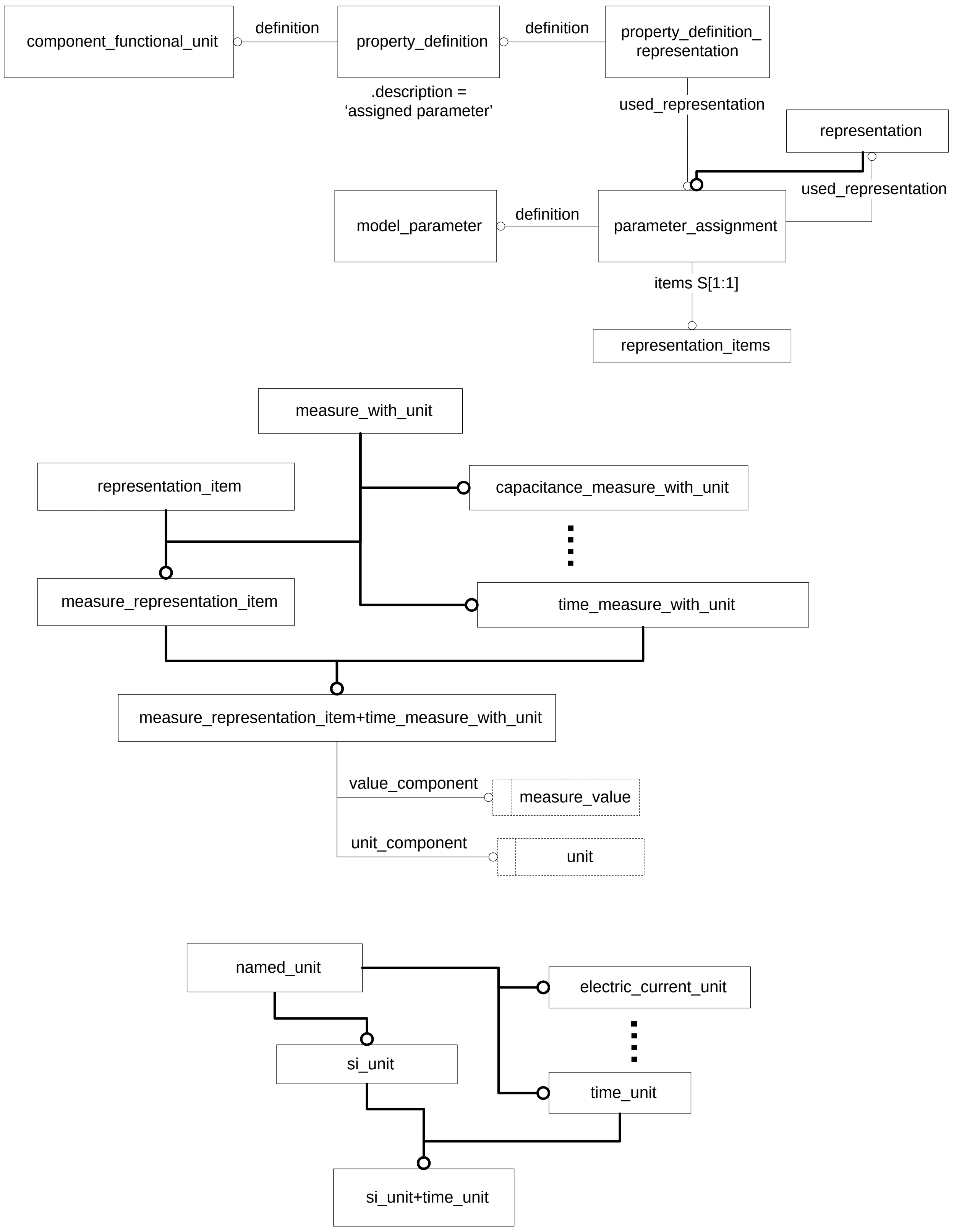


Figure 8. MIM mapping of Parameter_assignment and Numerical_item_with_unit.

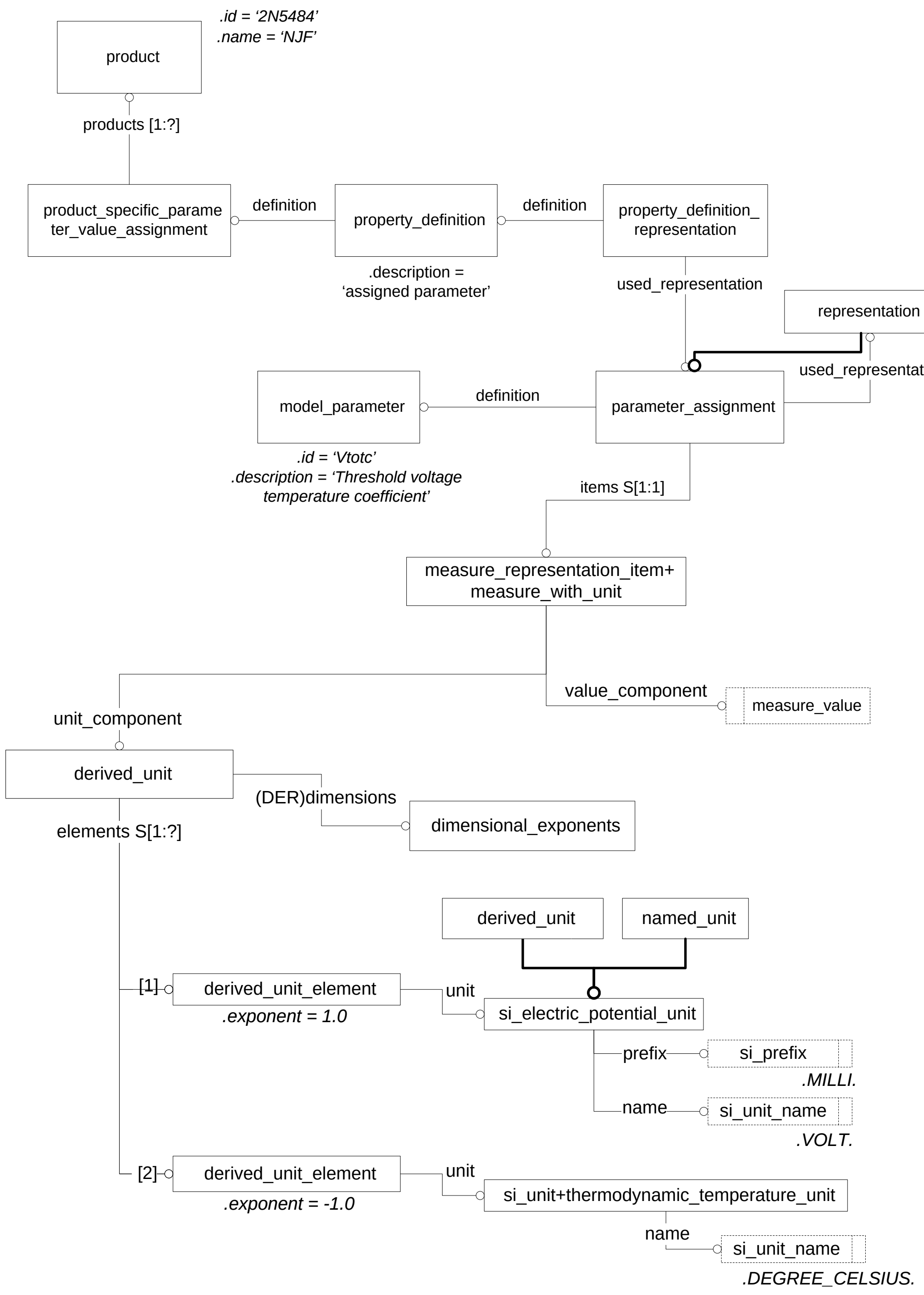
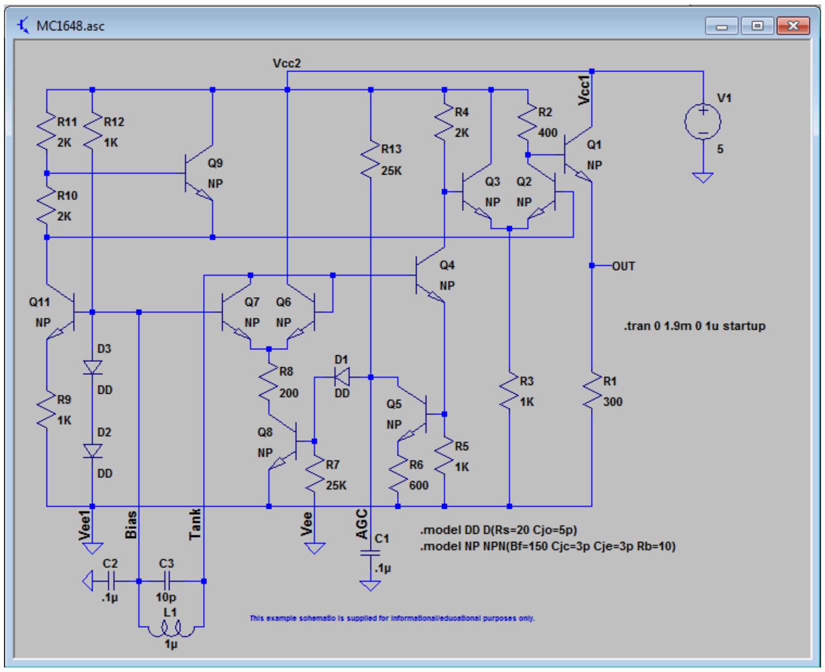


Figure 9. A representative parameter_assignment with a derived_unit associated with a product.

APPENDIX A: Representative SPICE netlists generated from AP210 functional models.

```
* LTSpiceIV Netlist Generated from AP210 file: MC1648.stp
C1 AGC 0 0.1µ
C2 Bias 0 0.1µ
C3 Tank Bias 10p
D1 AGC N007 DD
D2 N011 0 DD
D3 Bias N011 DD
L1 Tank Bias 1µ
Q1 Vcc1 N001 OUT 0 NP
Q11 N004 Bias N008 0 NP
Q2 N001 N004 N005 0 NP
Q3 Vcc1 N003 N005 0 NP
Q4 N003 Tank N010 0 NP
Q5 AGC N010 N012 0 NP
Q6 Vcc1 Tank N006 0 NP
Q7 Tank Bias N006 0 NP
Q8 N009 N007 0 0 NP
Q9 Vcc1 N002 N004 0 NP
R1 OUT 0 300
R10 N002 N004 2k
R11 Vcc1 N002 2k
R12 Vcc1 Bias 1k
R13 Vcc1 AGC 25k
R2 Vcc1 N001 400
R3 N005 0 1k
R4 Vcc1 N003 2k
R5 N010 0 1k
R6 N012 0 600
R7 N007 0 25k
R8 N006 N009 200
R9 N008 0 1k
V1 Vcc1 0 5
.model DD D(Cjo=5p Rs=20 )
.model NP NPN(Bf=150 Cjc=3p Cje=3p Rb=10 )
* Appended dot commands extracted from input file.
.tran 0 1.9m 0 1u startup
.backanno
.end
```

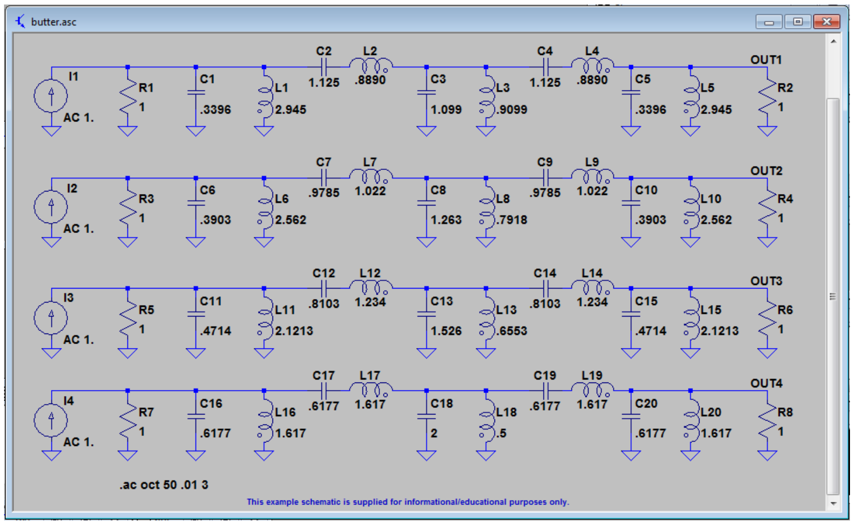


```
.tran 0 1.9m 0 1u startup
.model DD D(Rs=20 Cjo=5p)
.model NP NPN(Bf=150 Cjc=3p Cje=3p Rb=10)
```

This example schematic is supplied for informational/educational purposes only.

Figure A1. The LTspice netlist generated from the MC168.stp test case. The corresponding schematic is provided for reference.

```
* LTSpiceIV Netlist Generated from AP210 file: butter.stp
C1 N001 0 0.3396
C10 OUT2 0 0.3903
C11 N009 0 0.4714
C12 N010 N009 0.8103
C13 N011 0 1.526
C14 N012 N011 0.8103
C15 OUT3 0 0.4714
C16 N013 0 0.6177
C17 N014 N013 0.6177
C18 N015 0 2
C19 N016 N015 0.6177
C2 N002 N001 1.125
C20 OUT4 0 0.6177
C3 N003 0 1.099
C4 N004 N003 1.125
C5 OUT1 0 0.3396
C6 N005 0 0.3903
C7 N006 N005 0.9785
C8 N007 0 1.263
C9 N008 N007 0.9785
I1 0 N001 1 AC=1
I2 0 N005 1 AC=1
I3 0 N009 1 AC=1
I4 0 N013 1 AC=1
L1 N001 0 2.945
L10 OUT2 0 2.562
L11 N009 0 2.1213
L12 N010 N011 1.234
L13 N011 0 0.6553
L14 N012 OUT3 1.234
L15 OUT3 0 2.1213
L16 N013 0 1.617
L17 N014 N015 1.617
L18 N015 0 0.5
L19 N016 OUT4 1.617
L2 N002 N003 0.889
L20 OUT4 0 1.617
L3 N003 0 0.9099
L4 N004 OUT1 0.889
L5 OUT1 0 2.945
L6 N005 0 2.562
L7 N006 N007 1.022
L8 N007 0 0.7918
L9 N008 OUT2 1.022
R1 N001 0 1
R2 OUT1 0 1
R3 N005 0 1
R4 OUT2 0 1
R5 N009 0 1
R6 OUT3 0 1
R7 N013 0 1
R8 OUT4 0 1
* Appended dot commands extracted from input file.
.ac oct 50 .01 3
.backanno
.end
```


butter.asc
I1 AC 1. R1 1 C1 .3396 L1 2.945 C2 1.125 L2 .8890 C3 1.099 L3 .9099 C4 1.125 L4 .8890 C5 .3396 L5 2.945 OUT1 R2 1
I2 AC 1. R3 1 C6 .3903 L6 2.562 C7 .9785 L7 1.022 C8 1.263 L8 .7918 C9 .9785 L9 1.022 C10 .3903 L10 2.562 OUT2 R4 1
I3 AC 1. R5 1 C11 .4714 L11 2.1213 C12 .8103 L12 1.234 C13 1.526 L13 .6553 C14 .8103 L14 1.234 C15 .4714 L15 2.1213 OUT3 R6 1
I4 AC 1. R7 1 C16 .6177 L16 1.617 C17 .6177 L17 1.617 C18 2 L18 .5 C19 .6177 L19 1.617 C20 .6177 L20 1.617 OUT4 R8 1
.ac oct 50 .01 3

This example schematic is supplied for informational/educational purposes only.

Figure A2. The LTspice netlist generated from the butter.stp test case. The corresponding schematic is provided for reference.

```
* LTSpiceIV Netlist Generated from AP210 file: Clapp.stp
C1 N002 N003 750p
C2 N004 0 50p
C3 N003 0 750p
D2 0 N002 1N4148
J1 N001 N002 N003 2N5484
L1 N002 N004 100µ
L4 N003 0 1m Rser=100
V1 N001 0 10
.model 1N4148 D(Cjo=4p Iave=200m Is=2.52n M=0.4 N=1.752 Rs=0.568 Vpk=75 tt=20n )
.model 2N5484 NJF(Alpha=0.0001 Beta=3m Betatce=-0.5 Cgd=4p Cgs=4p Is=0.25p Kf=0
Lambda=10m Rd=10 Rs=10 Vk=80 Vto=-1.5 Vtotc=-3m )
* Appended dot commands extracted from input file.
.tran 250µ startup
.options method=trap
.backanno
.end
```

```
* LTSpiceIV Netlist Generated from AP210 file: divider_with_subckt.stp
C2 N004 0 50p
C3 N003 0 750p
V1 a 0 10
X1 a b 0 divider
X2 N003 b N004 divider
.subckt divider x1 x2 x3
r1 x1 x2 2k
r2 x2 x3 2k
.ends
* Appended dot commands extracted from input file.
.tran 3µ
.end
```

Figure A3. The LTspice netlists generated from the Clapp.stp and divider_with_subckt.stp test cases (see Figure 5 for the Clapp schematic).